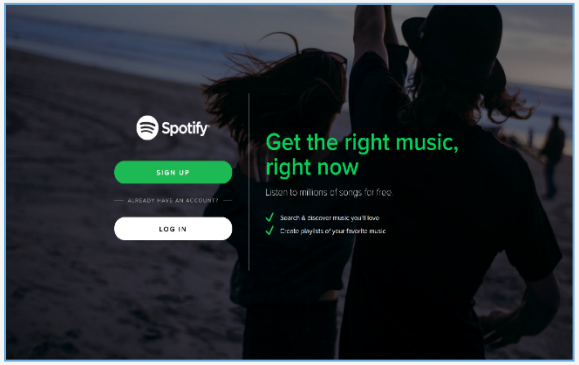
BLUE SPIKE

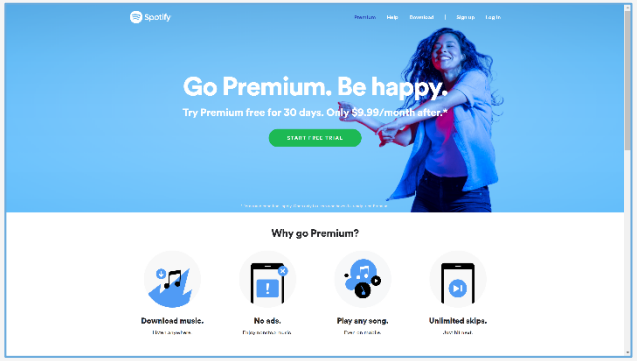
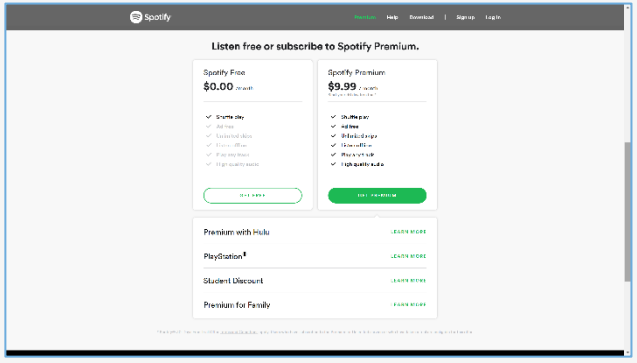
Ex. C. Infringement Chart for Claim 35 of the '011 Patent

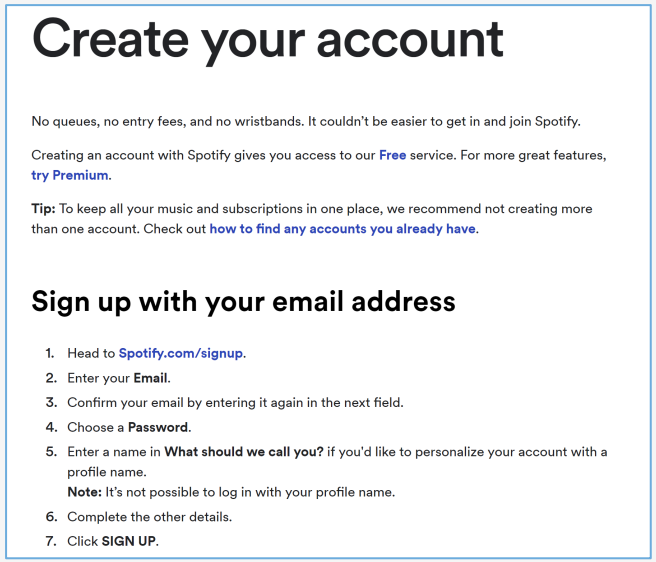
SPOTIFY

PATENT DETAILS

U.S. Patent No. 8,538,011	Systems, methods and devices for trusted transactions
ISSUE DATE : September 17, 2013 **FILING DATE :** December 7, 1999 **INVENTOR :** Scott A. Moskowitz	ABSTRACT: The invention discloses a system for enhancing trust in transactions, most particularly in remote transactions between a plurality of transactional parties, for instance a seller and buyer(s) of goods and/or services over a public computer network such as the internet. Trust is disclosed to be a multivalent commodity, in that the trust that is to be enhanced relates to information about the subject matter of the transactions (e.g., the suitability of the goods and services sold), the bona fides of the supplier of the goods and services, the appropriateness of a pricing structure for a particular transaction or series of transactions, a quantum of additional transactional value that may be imparted to the transactional relationship, security of information exchange, etc. An important contributor to trust for such aspects of the transaction is disclosed to be the use of highly-secure steganographic computer processing means for data identification, authentication, and transmission, such that confidence in the transaction components is enhanced. By providing an integrated multivalent system for enhancing trust across a variety of categories (for a variety of transaction species, including those in which the need for trust is greater on the part of one party than of another, as well as those in which both require substantial trust enhancement), the invention reduces barriers to forming and optimizing transactional relationships.

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	*Spotify offers a trusted transaction device through their premium media streaming services which can be played on any device. More specifically, Spotify uses a digital rights management system to encrypt the music content which is playable through the Spotify app only. For instance, to play the encrypted music on web browser, Spotify utilizes Encrypted Media Extensions. Spotify uses Google Widevine CDM to enable instant streaming on consumer devices, such as Chrome Browser on Desktop, Firefox Browser on Desktop, Chromecast, Android TVs etc.*   *See* Exhibit 1, p. 1, Spotify Web page (Date Accessed 04/27/2018), available at https://open.spotify.com/browse *See* Exhibit 2, "Discover and stream music across all your devices with Spotify _ FileHippo News" webpage (Date Accessed 04/27/2018), available at https://news.filehippo.com/2016/12/discover-and-stream-music-across-all-your-devices-with-spotify/

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	*Spotify offers Digital Rights Management to offer protection against illegal songs downloads through Encrypted Media Extensions (EME). It offers a freemium service to conduct trusted transactions between the user and the Spotify server. The subscription to Spotify allows playback of songs only through Web player or the application and cannot be accessed beyond them.*   *See* Exhibit 3, p. 1 & 2, "Spotify Premium" webpage (Date Accessed 04/30/2018), available at https://www.spotify.com/us/premium/

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	

See Exhibit 4, p. 1, "Create your account - Spotify" webpage (04/30/2018), available at https://support.spotify.com/us/account_payment_help/account_basics/create-your-spotify-account/

> "**Spotify also plays through several sound systems, TVs, and car stereo systems**. Please contact your device's manufacturer to determine whether it supports Spotify and, if so, how to enable it.
>
> Note: **The Spotify app can vary across different devices and operating systems**. We recommend running the latest firmware on your device for the best experience."

See Exhibit 5, p. 1, "System requirements - Spotify" webpage (04/30/2018), available at https://support.spotify.com/dk/article/spotify-system-requirements/ (emphasis added)

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	

System requirements

The following are the system requirements for using Spotify and accessing Spotify content through the **Spotify app**:

	Model	OS
iPhone	iPhone 4S or above	iOS 9 or above. 100 MB free space
iPad	iPad 2 or above	iOS 9 or above. 100 MB free space
iPod	5th generation iPod Touch or above	iOS 9 or above. 100 MB free space
Android	Any device	Android OS 4.1 or above. 500 MB free space
Mac	Any device	OS X 10.9 or above
Windows	Any device	Windows 7 or above

"**The web player**
The web player is supported by the following web browsers:

Chrome
Firefox
Edge
Opera

Spotify content may only be accessed with the Spotify app, the web player website, or apps otherwise authorized by Spotify."

See Exhibit 5, p. 1, "System requirements - Spotify" webpage (04/30/2018), available at https://support.spotify.com/dk/article/spotify-system-requirements/ (emphasis added)

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	*Upon information and belief, services like Spotify uses DRM for playback of the content.* "We **consume movies and music through services like** Netflix, **Spotify**, Pandora, Apple Music, etc., **often over the web**. **Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve**. That's where **EMEs come into play**. They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content** to the user. This means **Netflix can serve you a movie that's encrypted and protected by DRM**." *See* Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	*Upon information and belief, like Felix, the DRM implemented in Spotify works similarly.* "Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here's what happens: **The browser will load up this content, realize it is encrypted and then trigger some Javascript code**. This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later. **The CDM will then trigger a request that will be relayed by Felix to the license server**. This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM**. **The CDM will then decrypt the content and it will begin playing for the user**." The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that's trusted by Felix**. If it isn't, Felix can't be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google's Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**. *See* Exhibit 6, p. 2 & 3, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	"Web Playback SDK Quick Start BETA Create a **simple web app with the Web Playback SDK that allows you to play an audio track inside your browser**. By using Spotify developer tools, you accept our Developer Terms of Service. They contain important information about what you can and can't do with our developer tools. Please read them carefully. Introduction **The Web Playback SDK is client-side JavaScript library which allows you to create a new player in Spotify Connect and play any audio track from Spotify in the browser via Encrypted Media Extensions**. You can read more about the SDK in the overview, or dig into the reference documentation. In this Quick Start, we will be adding the Web Playback SDK to a simple HTML page. Authenticating with Spotify **You will need an access token from your personal Spotify Premium account. Click the button below to quickly obtain your access token**." *See* Exhibit 7, p. 1, "Web Playback SDK Quick Start _ Spotify for Developers" webpage (Date Accessed 05/01/2018), available at https://beta.developer.spotify.com/documentation/web-playback-sdk/quick-start/ (emphasis added)

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	**"Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video**. EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system**. The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance "EME implementations use the following external components: **Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below)**. **Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available**. **License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application**. *continued on next slide.*

CLAIM 35	SPOTIFY
35. A device for conducting trusted transactions between at least two parties, comprising:	**Packaging service: Encodes and encrypts media for distribution/consumption**. Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application**: **when a user signs into the application, the application determines the user's identity and privileges**." *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

CLAIM 35	SPOTIFY
a steganographic cipher;	*The data is encrypted using a steganographic cipher (here **Encrypted Media Extensions (EME)**) before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key.* "We **consume movies and music through services like** Netflix, **Spotify**, Pandora, Apple Music, etc., **often over the web**. **Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve**. That's where **EMEs come into play**. They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content** to the user. This means **Netflix can serve you a movie that's encrypted and protected by DRM**." *See* Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
a steganographic cipher;	"Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here's what happens: **The browser will load up this content, realize it is encrypted and then trigger some Javascript code**. This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later. **The CDM will then trigger a request that will be relayed by Felix to the license server**. This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM**. **The CDM will then decrypt the content and it will begin playing for the user**." The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that's trusted by Felix**. If it isn't, Felix can't be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google's Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**. *See* Exhibit 6, p. 2 & 3, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
a steganographic cipher;	**"Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video**. EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system**. The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance "EME implementations use the following external components: **Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below)**. **Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available**. **License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application**. *continued on next slide.*

CLAIM 35	SPOTIFY
a steganographic cipher;	**Packaging service: Encodes and encrypts media for distribution/consumption**. Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application**: **when a user signs into the application, the application determines the user's identity and privileges**." *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added) "The **authorization flow we use in this tutorial is the Authorization Code Flow**. **This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token**. The **code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication**." *See* Exhibit 9, p. 1, "Web API Tutorial _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/web-api/quick-start/ (emphasis added)

CLAIM 35	SPOTIFY
a steganographic cipher;	"How does EME Work?"

1. A web application attempts to play audio or video that has one or more encrypted streams.
2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption.
3. The application handles the encrypted event:
    a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server.
    b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding.
4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s).
5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM, by calling generateRequest() on the MediaKeySession.
6. The CDM fires a message event: a request to acquire a key from a license server.
7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example).
8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession.
9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID.

Media playback resumes.

How does the browser know that media is encrypted?

This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry.

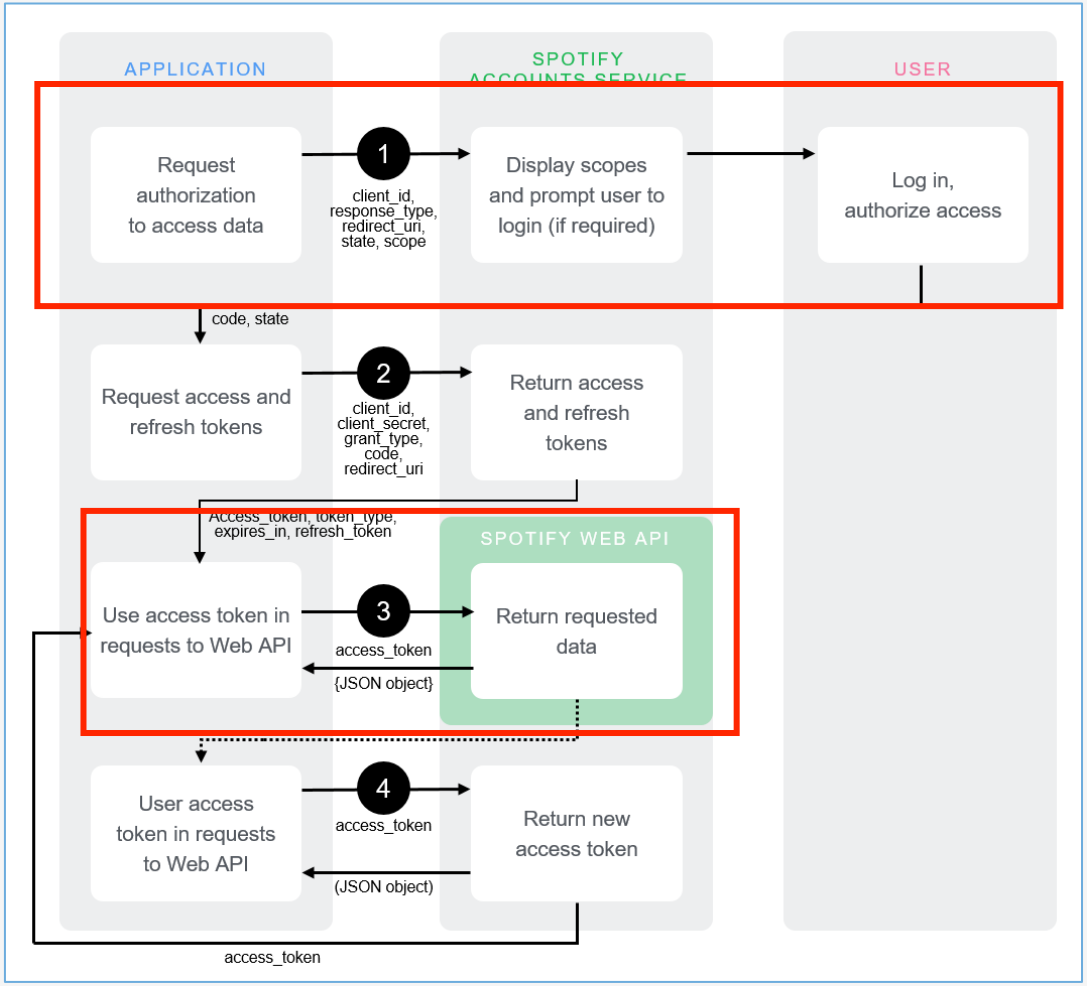
Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license.

See Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added)

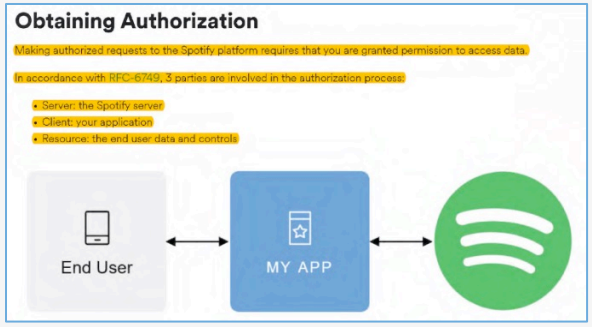
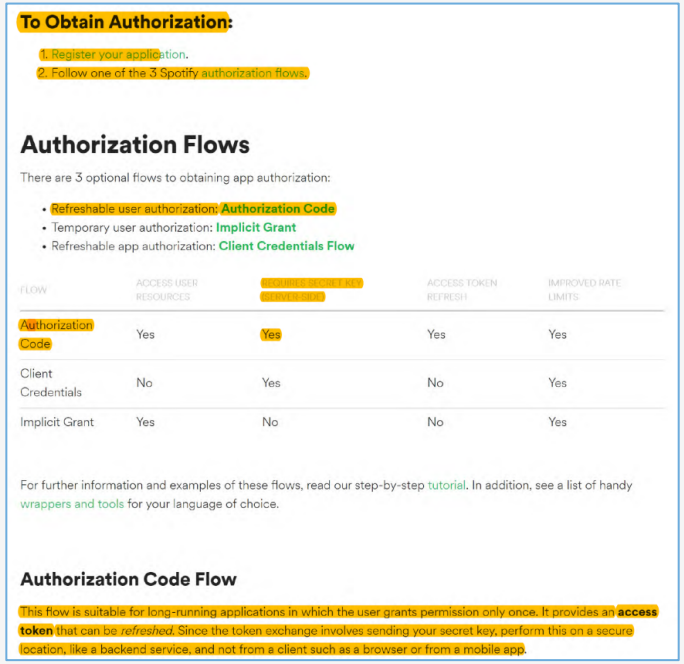
Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.

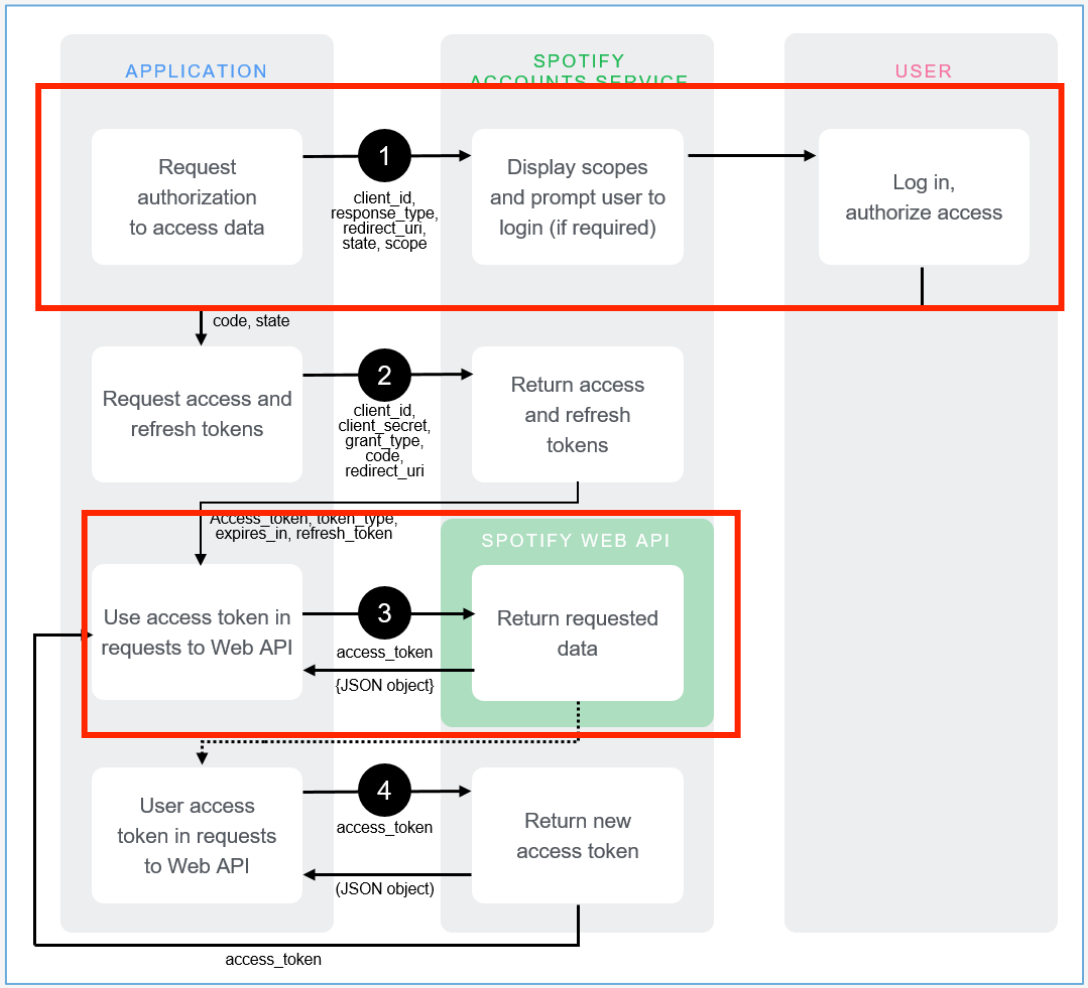
CLAIM 35	SPOTIFY
a controller for receiving input data or outputting output data; and	*Upon information and belief, Spotify SDK offers a controller for receiving the input data such as user credentials for authentication and if the authentication happens, it outputs the validation by redirecting to the application with access token.* "There are **two basic ways you can authenticate your application's user and pass the scopes needed to get authorization to access user data**: The preferred and highly recommended way: **Use the Spotify client to authenticate the user (with a fallback to login through a WebView).** If Spotify is installed on the device, **SDK connects to the Spotify client and uses current session. If Spotify is not installed, this method will fall back to using Android WebView class which allows you to display a web page as part of your activity layout**. **Authentication and authorization takes place in the WebView without leaving the application**. Only if the first way is not possible: **Login through a web browser. This method opens the Spotify Accounts page in a separate, external browser window, completes the authentication process, and then redirects back to your application**." *See* Exhibit 10, p. 1, "Android SDK Authentication Guide _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/ (emphasis added)

CLAIM 35	SPOTIFY
a controller for receiving input data or outputting output data; and	""Single **Sign-On with Spotify Client** and a WebView Fallback In this flow, **the Android SDK will try to fetch the authorization code/access token using the Spotify Android client**. Note: to be able to use Single Sign-On you need to register your application's fingerprint. Please see Registering Application Fingerprint section of the tutorial. **If Spotify is installed on the device, the SDK will connect to the Spotify client and fetch the authorization code/access token for current user.** Since the user is already logged into Spotify, they don't need to type in their username and password. If the SDK application requests scopes that have not been approved before, the user will see a list of scopes and will need to accept them. If **Spotify is not installed on the device, the SDK will fallback to the WebView based authorization and open the Spotify Accounts login page at https://accounts.spotify.com in a native WebView**. User will have to enter their username and password to login to Spotify and accept the supplied scopes. In both cases **the result of the authorization flow will be returned in the onActivityResult method of the activity that initiated it**. This flow is entirely completed within the application; there is no need to open a web browser." *See* Exhibit 10, p. 2, "Android SDK Authentication Guide _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/ (emphasis added)

CLAIM 35	SPOTIFY
a controller for receiving input data or outputting output data; and	

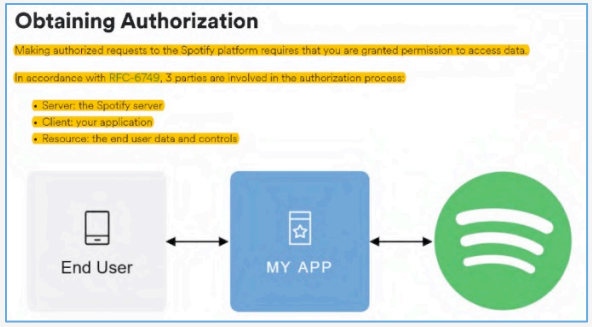
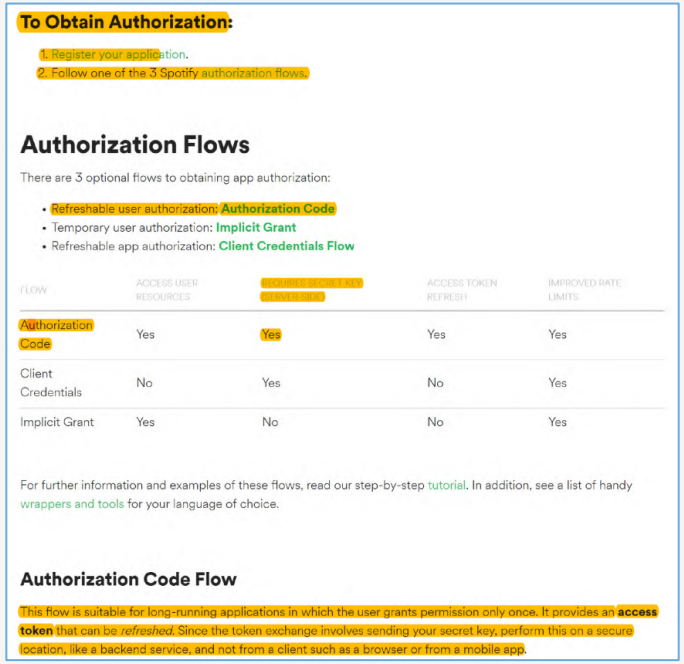
See Exhibit 11, p. 4, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added)

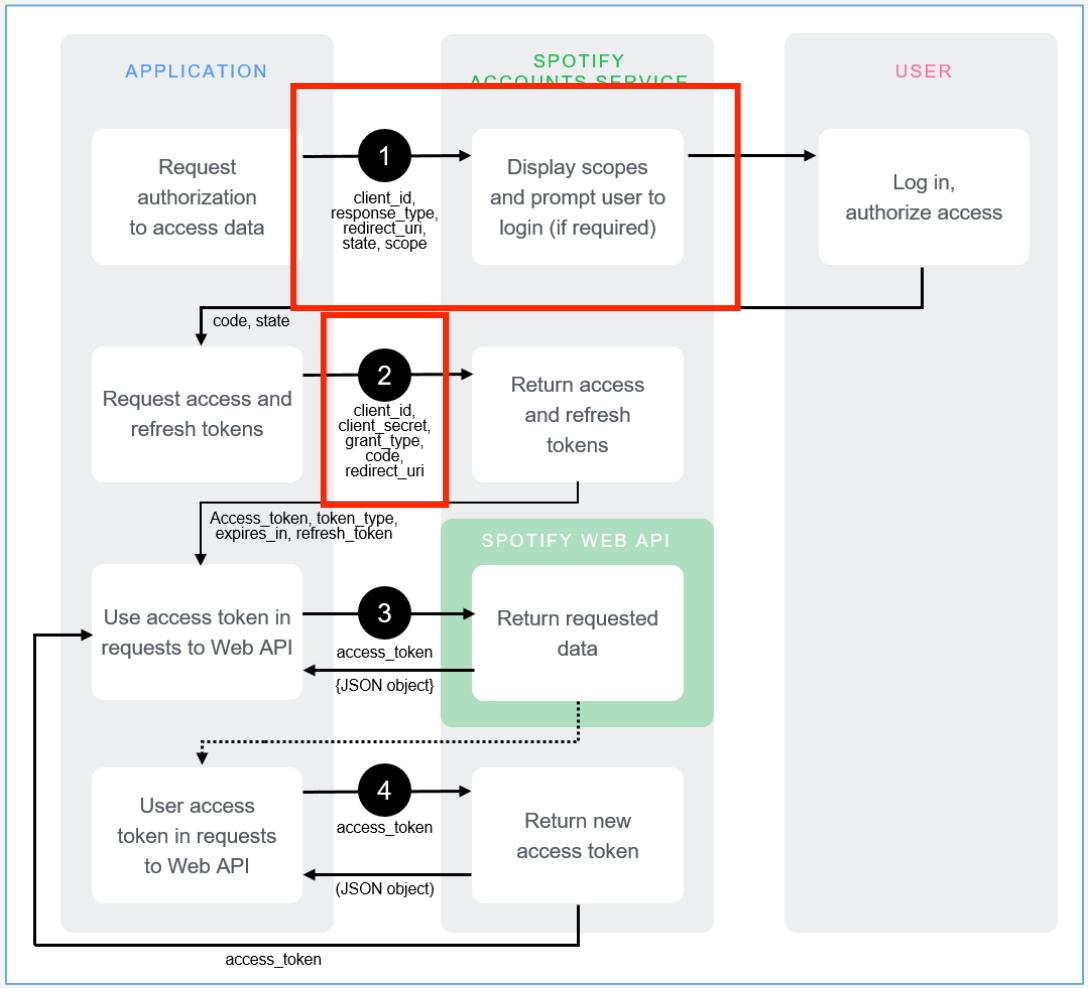
CLAIM 35	SPOTIFY
a controller for receiving input data or outputting output data; and	  *See* Exhibit 11, p. 1-3, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

CLAIM 35	SPOTIFY
at least one input/output connection,	 *See* Exhibit 11, p. 4, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added)

CLAIM 35	SPOTIFY
at least one input/output connection,	Spotify (Encrypted Media Extension) system offers an input/output connection for inputting a user credentials and accordingly outputting the audio based on the response. "Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here's what happens: **The browser will load up this content, realize it is encrypted and then trigger some Javascript code**. This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later. **The CDM will then trigger a request that will be relayed by Felix to the license server**. This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM**. **The CDM will then decrypt the content and it will begin playing for the user**." The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that's trusted by Felix**. If it isn't, Felix can't be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google's Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**. *See* Exhibit 6, p. 2 & 3, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
at least one input/output connection,	*The data is encrypted using a steganographic cipher (here **Encrypted Media Extensions (EME)**) before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key.* "We **consume movies and music through services like** Netflix, **Spotify**, Pandora, Apple Music, etc., **often over the web**. **Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve**. That's where **EMEs come into play**. They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content** to the user. This means **Netflix can serve you a movie that's encrypted and protected by DRM**." *See* Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	  *See* Exhibit 11, p. 1-3, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added)

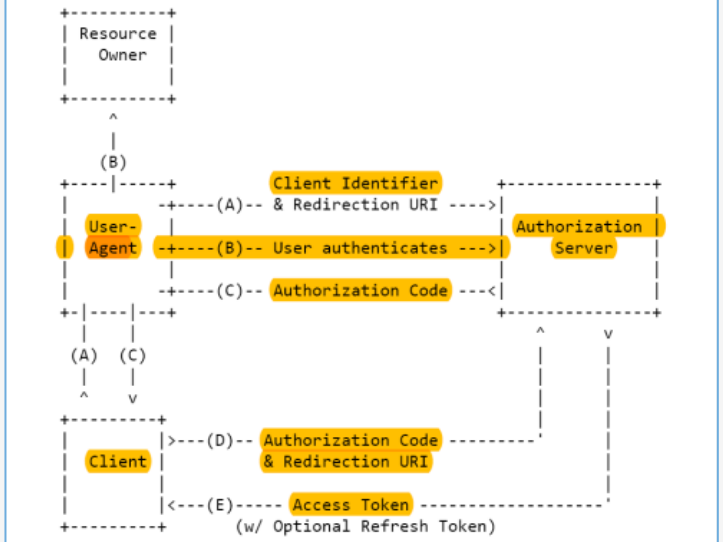
CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	

See Exhibit 11, p. 4, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added)

CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	*Spotify complies with RFC 6749 standard for authorization code flow*

4.1. Authorization Code Grant

The authorization code grant type is used to obtain both access tokens and refresh tokens and is optimized for confidential clients. Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.



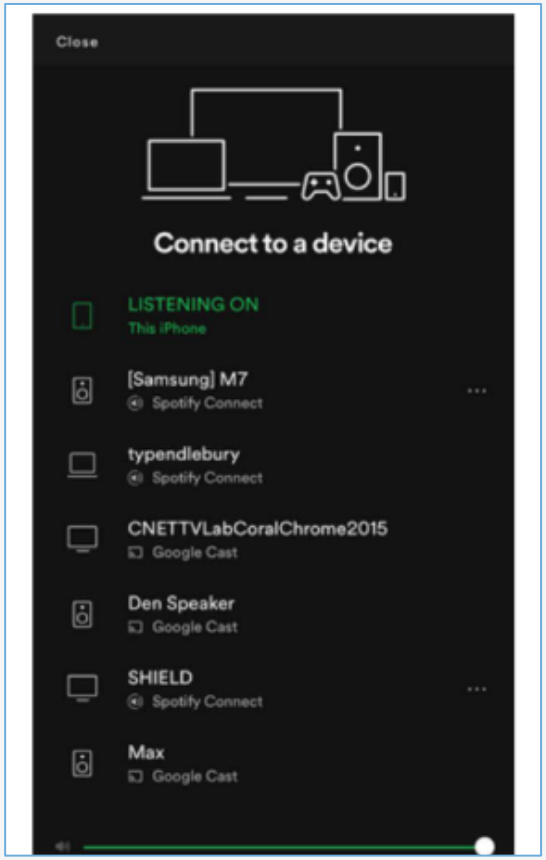
Note: The lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent.

Figure 3: Authorization Code Flow

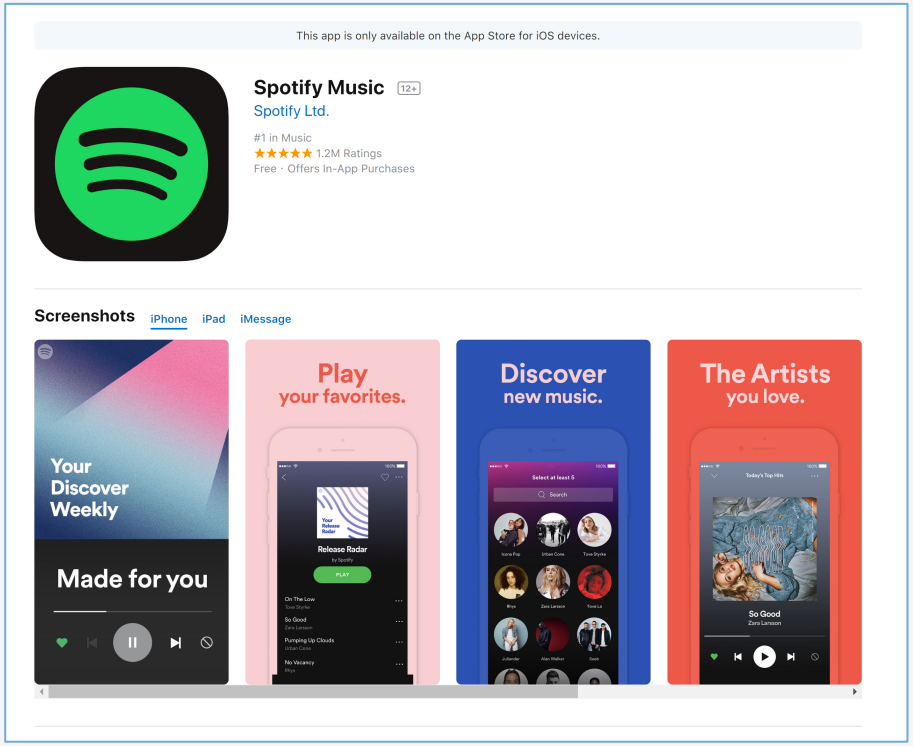
See Exhibit 14, p. 24, "RFC 6749 - The OAuth 2.0 Authorization Framework" webpage (Date Accessed 05/01/2018), available at https://tools.ietf.org/html/rfc6749#section-4.1 (emphasis added)

CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	"The flow illustrated in Figure 3 includes the following steps: (A) The **client initiates the flow by directing the resource owner's user-agent to the authorization endpoint**. The **client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted** (or denied). (B) The **authorization server authenticates the resource owner** (via the user-agent) and establishes whether the resource owner grants or denies the client's access request. (C) Assuming the resource owner grants access, **the authorization server redirects the user-agent back to the client using the redirection URI provided earlier** (in the request or during client registration). The redirection URI includes an authorization code and any local state provided by the client earlier. (D) The **client requests an access token from the authorization server's token endpoint by including the authorization code received in the previous** step. When making the request, the client authenticates with the authorization server. **The client includes the redirection URI used to obtain the authorization code for verification**. (E) The **authorization server authenticates the client, validates the authorization code, and ensures that the redirection URI received matches the URI used to redirect the client in step (C)**. If valid, **the authorization server responds back with an access token** and, optionally, a refresh token." *See* Exhibit 14, p. 25, "RFC 6749 - The OAuth 2.0 Authorization Framework" webpage (Date Accessed 05/01/2018), available at https://tools.ietf.org/html/rfc6749#section-4.1 (emphasis added)

CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	"How to use The **easiest way to use Spotify Connect is with the Android or iOS app**. You don't have to perform a set up routine to stream to a new speaker (as you would with most multi-room services); if the **Spotify Connect device is on the network, your phone should be able to see it**. Once the app detects compatible speakers, a **"Devices Available" message appears at the bottom of the screen**. Pressing this will **bring up a list of the players on your network, with a choice of Spotify Connect, Chromecast or even AirPlay/Bluetooth connections**. **Once you choose a player to stream to, your music will begin playing. If you have a speaker group created within Google Home, you can now stream to all of them together using Spotify as well**." "Keep in mind that **all compatible devices on the network, plus ones you have previously logged into around the world, will appear in the list**. This means if you really want to mess with your pets at home, you can." *See* Exhibit 21, p. 5 & 6, "Spotify Connect_ What it is and how it works - CNET" webpage (Date Accessed 05/03/2018), available at https://www.cnet.com/news/spotify-connect-what-is-and-how-it-works/ (emphasis added)

CLAIM 35	SPOTIFY
wherein the device has a device identification code stored in the device;	 *See* Exhibit 21, p. 6, "Spotify Connect_ What it is and how it works - CNET" webpage (Date Accessed 05/03/2018), available at https://www.cnet.com/news/spotify-connect-what-is-and-how-it-works/ *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

CLAIM 35	SPOTIFY
a steganographically ciphered software application;	*Spotify Application (Android, iOS, Windows) is a steganographically ciphered software application which encrypts the songs and are playable only through the application.*



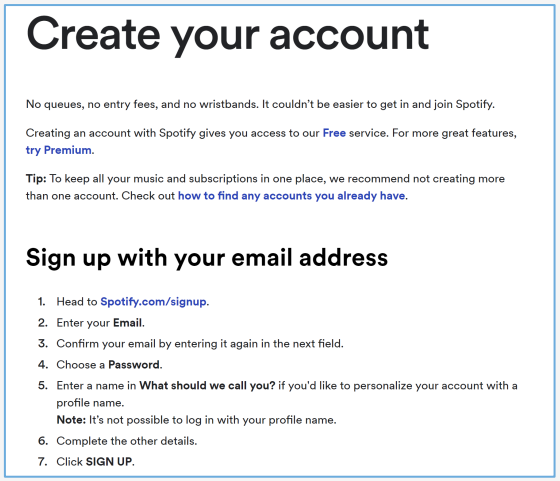
See Exhibit 22, p. 1, "Spotify Music on the App Store" webpage (Date Accessed 05/03/2018), available at https://itunes.apple.com/us/app/spotify-music/id324684580?mt=8

CLAIM 35	SPOTIFY
a steganographically ciphered software application;	Description **Spotify is the best way to listen to music on mobile or tablet.** Search for any track, artist or album and listen for free. Make and share playlists. Build your biggest, best ever music collection. Get inspired with personal recommendations, and readymade playlists for just about everything. **Listen absolutely free with ads, or get Spotify Premium**. Free on mobile • Play any artist, album, or playlist in shuffle mode. Free on tablet • Play any song, any time. **Premium features** **• Play any song, any time on any device: mobile, tablet or computer.** **• Enjoy ad-free music.** **• Listen offline.** **• Get better sound quality**. *See* Exhibit 22, p. 1, "Spotify Music on the App Store" webpage (Date Accessed 05/03/2018), available at https://itunes.apple.com/us/app/spotify-music/id324684580?mt=8 (emphasis added)

CLAIM 35	SPOTIFY
a steganographically ciphered software application;	Can you download music from Spotify? Yes and no. With **Spotify Premium you can set music to be available "offline" but it's not the same as downloading music in the traditional sense**. For example, you can't try to game the system by downloading an album then cancelling your subscription at a later date. And you can't download the tracks to burn them to a CD or copy them to other devices. The idea of Spotify's offline mode is to allow you to have access to your favorite music when you're trying to save mobile data or travelling somewhere where access to the internet might not be easy. **With Spotify Premium you can have up to 3,333 songs available to listen to offline on up to three different devices. Downloading songs, albums or playlists on Spotify is simple too, which is great. Just click "save" on the album you'd like to download to listen to it offline**. Alternatively, click the three dots next to a song and click "Save to Your Music". *See* Exhibit 15, p. 3, "What is Spotify and how does it work_ - Pocket-lint" webpage (Date Accessed 05/04/2018), available at https://www.pocket-lint.com/apps/news/spotify/139236-what-is-spotify-and-how-does-it-work (emphasis added)

CLAIM 35	SPOTIFY
a steganographically ciphered software application;	Description **Spotify is the best way to listen to music on mobile or tablet.** Search for any track, artist or album and listen for free. Make and share playlists. Build your biggest, best ever music collection. Get inspired with personal recommendations, and readymade playlists for just about everything. **Listen absolutely free with ads, or get Spotify Premium**. Free on mobile • Play any artist, album, or playlist in shuffle mode. Free on tablet • Play any song, any time. **Premium features** **• Play any song, any time on any device: mobile, tablet or computer.** **• Enjoy ad-free music.** **• Listen offline.** **• Get better sound quality**. *See* Exhibit 22, p. 1, "Spotify Music on the App Store" webpage (Date Accessed 05/03/2018), available at https://itunes.apple.com/us/app/spotify-music/id324684580?mt=8 (emphasis added)

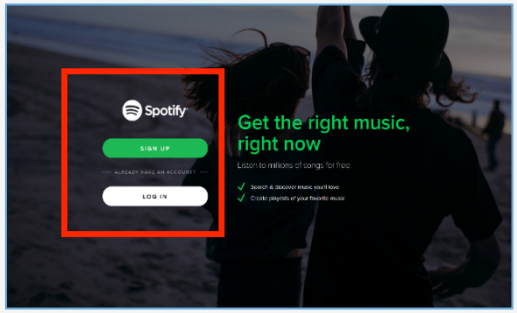
CLAIM 35	SPOTIFY
a steganographically ciphered software application;	"**Offline playlists are a Premium feature that allows you to download tracks onto your desktop or mobile device to play on a Spotify application**. However, this is somewhat different to purchased tracks. "Offline" allows you to store 3,333 tracks. **These are stored in an encrypted format, that the Spotify application then decodes and plays as some lovely music**. Purchasing tracks from our Download Store will allow you to have DRM mp3's on your computer, allowing you to download it to 3-5 computers (depending on the record label). You can find out more information about our download store here - and more about Offline Playlists here" *See* Exhibit 16, p. 2, "What is 'offline' vs purchased music - The Spotify Community" webpage (Date Accessed 05/03/2018), available at https://community.spotify.com/t5/Accounts/What-is-offline-vs-purchased-music/td-p/17603 (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

CLAIM 35	SPOTIFY
wherein said steganographically ciphered software application has been subject to a steganographic cipher for serialization;	*Spotify offers user login credentials to offer steganographically ciphered Spotify application having library of encrypted songs which can be accessed by the user after entering their personalization information.* 

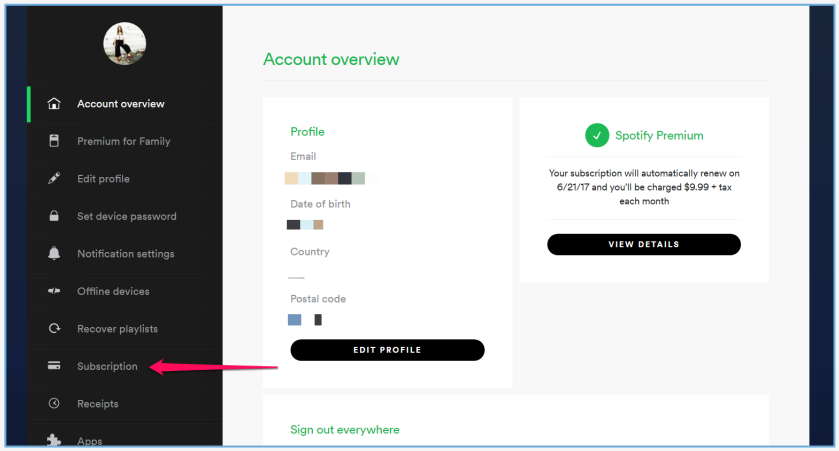
See Exhibit 4, p. 1, "Create your account - Spotify" webpage (04/30/2018), available at https://support.spotify.com/us/account_payment_help/account_basics/create-your-spotify-account/

> "**Spotify also plays through several sound systems, TVs, and car stereo systems**. Please contact your device's manufacturer to determine whether it supports Spotify and, if so, how to enable it.
>
> Note: **The Spotify app can vary across different devices and operating systems**. We recommend running the latest firmware on your device for the best experience."

See Exhibit 5, p. 1, "System requirements - Spotify" webpage (04/30/2018), available at https://support.spotify.com/dk/article/spotify-system-requirements/ (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographically ciphered software application has been subject to a steganographic cipher for serialization;	 *See* Exhibit 1, p. 1, Spotify Web page (Date Accessed 04/27/2018), available at https://open.spotify.com/browse "There are **two basic ways you can authenticate your application's user and pass the scopes needed to get authorization to access user data**: The preferred and highly recommended way: **Use the Spotify client to authenticate the user (with a fallback to login through a WebView).** If Spotify is installed on the device, **SDK connects to the Spotify client and uses current session. If Spotify is not installed, this method will fall back to using Android WebView class which allows you to display a web page as part of your activity layout**. **Authentication and authorization takes place in the WebView without leaving the application**. Only if the first way is not possible: **Login through a web browser. This method opens the Spotify Accounts page in a separate, external browser window, completes the authentication process, and then redirects back to your application**." *See* Exhibit 10, p. 1, "Android SDK Authentication Guide _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/ (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographically ciphered software application has been subject to a steganographic cipher for serialization;	""Single **Sign-On with Spotify Client** and a WebView Fallback In this flow, **the Android SDK will try to fetch the authorization code/access token using the Spotify Android client**. Note: to be able to use Single Sign-On you need to register your application's fingerprint. Please see Registering Application Fingerprint section of the tutorial. **If Spotify is installed on the device, the SDK will connect to the Spotify client and fetch the authorization code/access token for current user.** Since the user is already logged into Spotify, they don't need to type in their username and password. If the SDK application requests scopes that have not been approved before, the user will see a list of scopes and will need to accept them. If **Spotify is not installed on the device, the SDK will fallback to the WebView based authorization and open the Spotify Accounts login page at https://accounts.spotify.com in a native WebView**. User will have to enter their username and password to login to Spotify and accept the supplied scopes. In both cases **the result of the authorization flow will be returned in the onActivityResult method of the activity that initiated it**. This flow is entirely completed within the application; there is no need to open a web browser." *See* Exhibit 10, p. 2, "Android SDK Authentication Guide _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/android-sdk/guides/android-authentication/ (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographically ciphered software application has been subject to a steganographic cipher for serialization;	 *See* Exhibit 17, p. 2, "How to Cancel Spotify Premium_ A Step-by-Step Guide" webpage (Date Accessed 05/03/2018), available at https://www.dailydot.com/upstream/cancel-spotify-premium/ "3.1 Our Services & **Paid Subscriptions** **Spotify provides streaming services offering a selection of music and other content**. Certain Spotify services are provided to you free-of-charge. Other Spotify services require payment before you can access them. **The Spotify services that may be accessed after payment are currently referred to as the "Premium Service" and the "Unlimited Service" (together, the "Paid Subscriptions").** The Spotify service that does not require payment is currently referred to as the "Free Service". You can learn more about our services by visiting our website." *See* Exhibit 18, p. 4, "Terms and Conditions of Use - Spotify" webpage (Date Accessed 05/03/2018), available at https://www.spotify.com/is/legal/end-user-agreement/ (emphasis added)

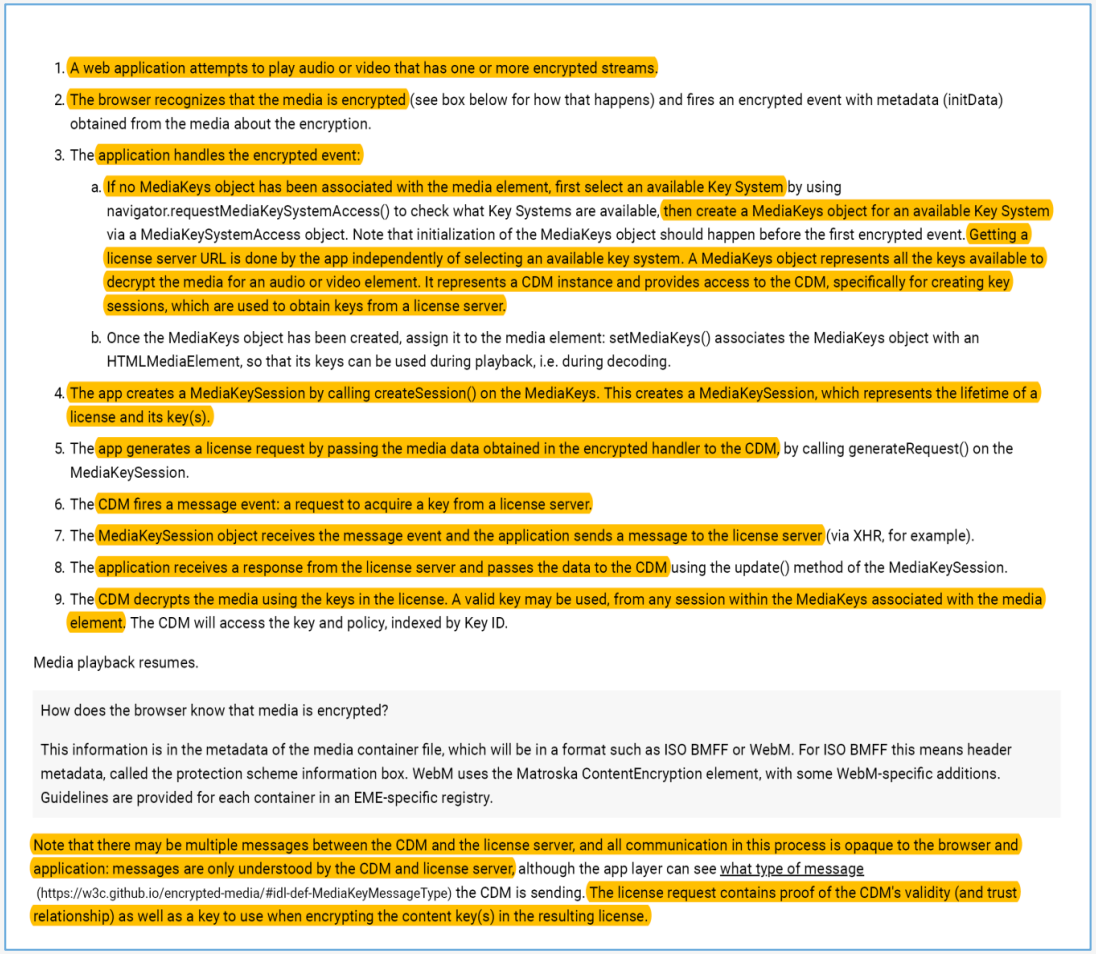
CLAIM 35	SPOTIFY
wherein said steganographically ciphered software application has been subject to a steganographic cipher for serialization;	"15 Payments, cancellations, and cooling off **Paid Subscriptions can be purchased either by (1) paying a monthly subscription fee**; or (2) pre-payment giving you access to the Spotify Service for a specific time period ("Pre-Paid Period"). **When you register for a Paid Subscription, Trial, or Code online, you consent to get access to Spotify Premium immediately**. If you reside outside the United States and register for a Paid Subscription or Code online, you may change your mind for any or no reason and receive a full refund of all monies paid within fourteen (14) days (the "Cooling-off Period"). Refunds will not, however, be provided if you have accessed Spotify at any time during the Cooling-off Period. Unless your **Paid Subscription has been purchased as a Pre-Paid Period, your payment to Spotify (or to a third party through whom you purchased the Paid Subscription, such as a telephone company) will automatically renew at the end of the subscription period**, unless you cancel your Paid Subscription through your subscription page before the end of the current subscription period." *See* Exhibit 18, p. 12 & 13, "Terms and Conditions of Use - Spotify" webpage (Date Accessed 05/03/2018), available at https://www.spotify.com/is/legal/end-user-agreement/ (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

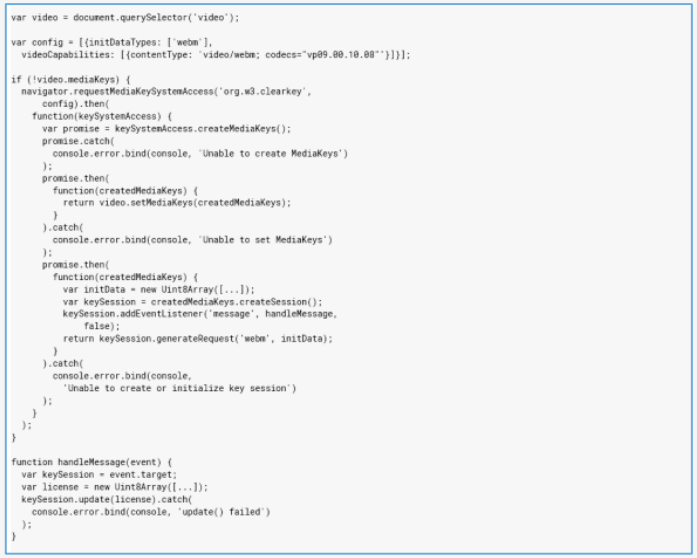
CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	*Spotify encrypts the data using a steganographic cipher (here **Encrypted Media Extensions (EME)**) in the App or web player before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key.* "We **consume movies and music through services like** Netflix, **Spotify**, Pandora, Apple Music, etc., **often over the web**. **Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve**. That's where **EMEs come into play**. They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content** to the user. This means **Netflix can serve you a movie that's encrypted and protected by DRM**." *See* Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

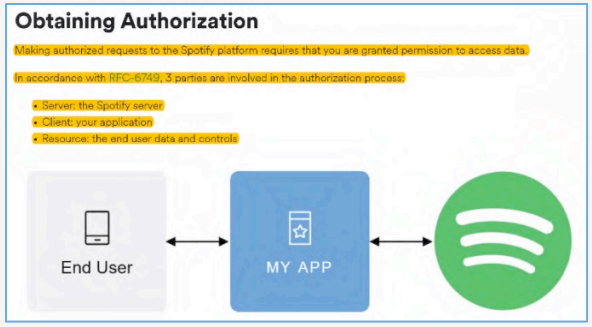
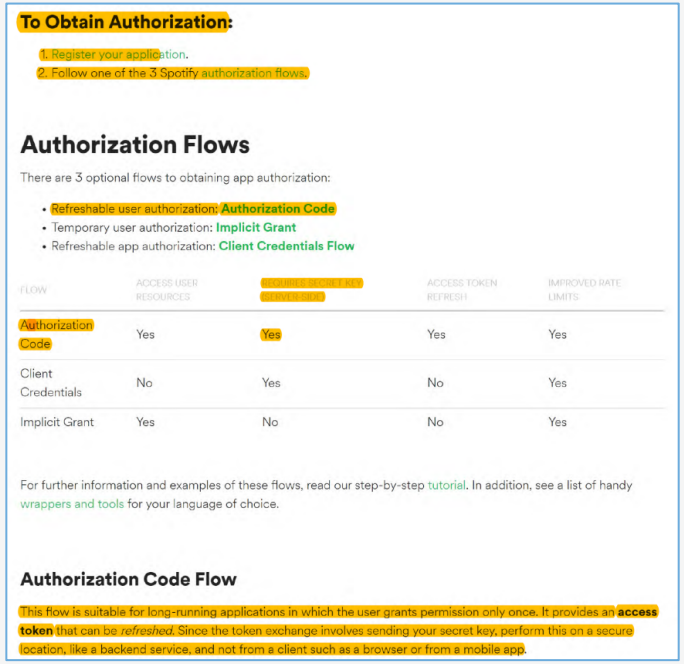
CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	"Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here's what happens: **The browser will load up this content, realize it is encrypted and then trigger some Javascript code**. This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later. **The CDM will then trigger a request that will be relayed by Felix to the license server**. This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM**. **The CDM will then decrypt the content and it will begin playing for the user**." The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that's trusted by Felix**. If it isn't, Felix can't be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google's Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**. *See* Exhibit 6, p. 2 & 3, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	**"Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video**. EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system**. The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance "EME implementations use the following external components: **Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below)**. **Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available**. **License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application**. *continued on next slide.*

CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	**Packaging service: Encodes and encrypts media for distribution/consumption**. Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application**: **when a user signs into the application, the application determines the user's identity and privileges**." *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added) "The **authorization flow we use in this tutorial is the Authorization Code Flow**. **This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token**. The **code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication**." *See* Exhibit 9, p. 1, "Web API Tutorial _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/web-api/quick-start/ (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	"How does EME Work?"  1. A web application attempts to play audio or video that has one or more encrypted streams. 2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption. 3. The application handles the encrypted event: a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server. b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding. 4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s). 5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM, by calling generateRequest() on the MediaKeySession. 6. The CDM fires a message event: a request to acquire a key from a license server. 7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example). 8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession. 9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID. Media playback resumes. How does the browser know that media is encrypted? This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry. Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license. *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added)

CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	*Upon information and belief, Spotify works similarly utilizing the CDM module as explained here.* "**Getting a key from a license server** In typical commercial use, **content will be encrypted and encoded using a packaging service or tool**. **Once the encrypted media is made available online, a web client can obtain a key (contained within a license) from a license server and use the key to enable decryption and playback of the content**. The following code (adapted from the spec examples) shows **how an application can select an appropriate key system and obtain a key from a license server**."  *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added)

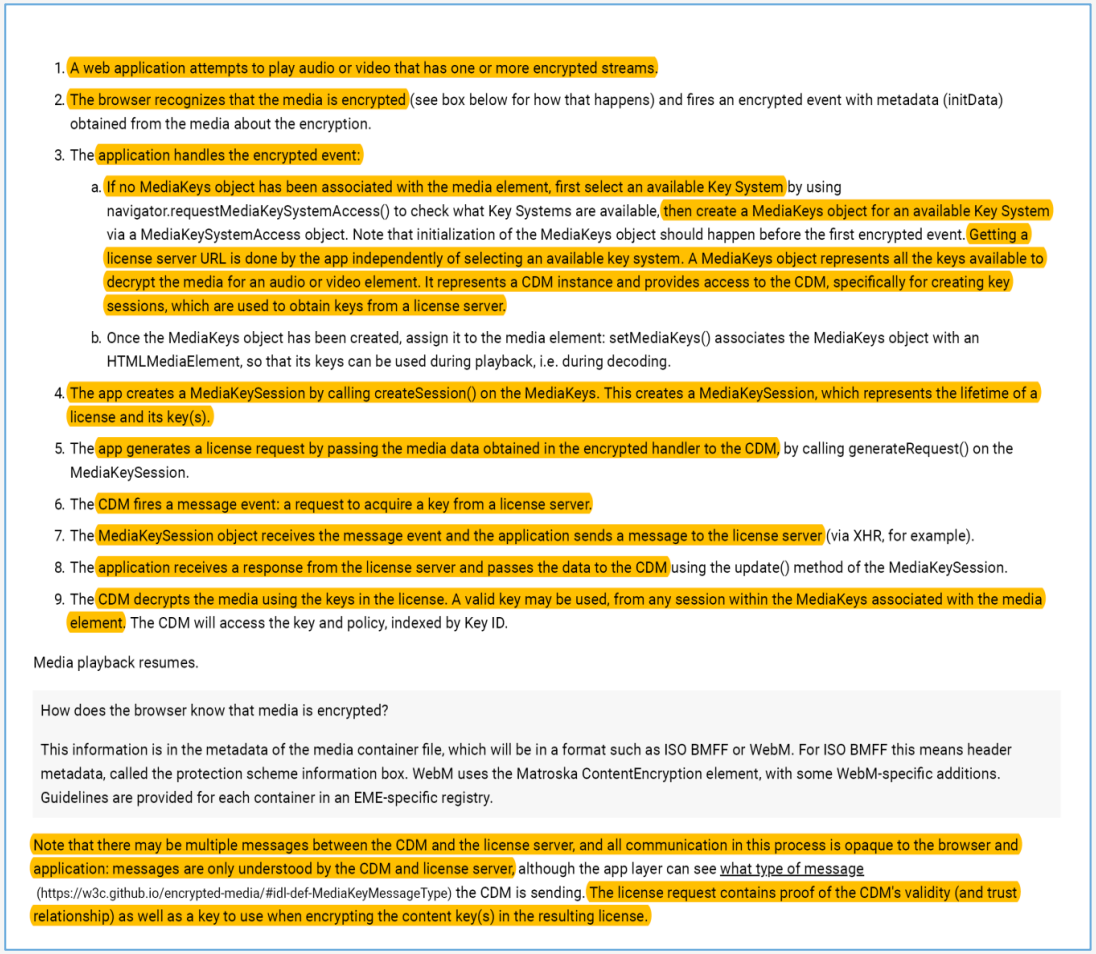
CLAIM 35	SPOTIFY
wherein said steganographic cipher receives said output data, steganographically ciphering said output data using a key, to define steganographically ciphered output data, and transmits said steganographically ciphered output data to said at least one input/ output connection;	  *See* Exhibit 11, p. 1-3, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*

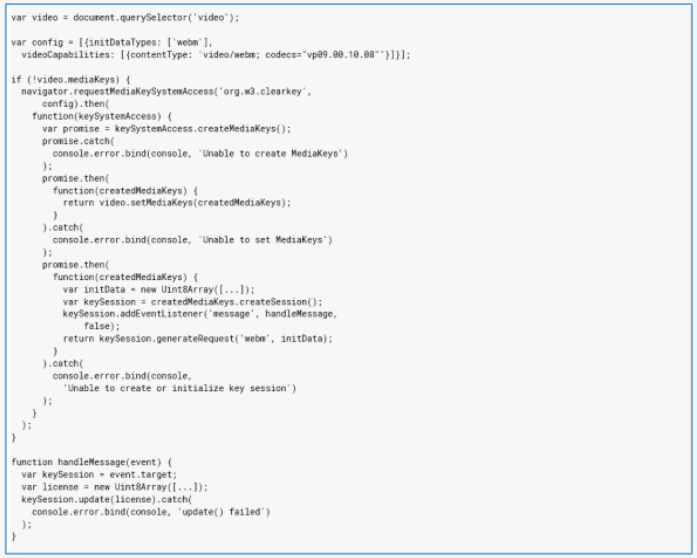
CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	*Spotify encrypts the data using a steganographic cipher (here **Encrypted Media Extensions (EME)**) in the App or web player before transmission from Party 1 to Party 2 so that it is not compromised or deciphered by an unauthorized user. The package's encrypted content will have a unique license identification information which can be only decrypted using a proper key. Further, the package's encrypted content includes audio content (value-added information) as well as metadata and headers that allow Spotify to display album art, lyrics, promotional material, etc. (at least one value-added component).* "We **consume movies and music through services like** Netflix, **Spotify**, Pandora, Apple Music, etc., **often over the web**. **Netflix, et al. are obligated, due to agreements with content partners, to serve their content with DRM.** Moreover, web streaming is likely a significant chunk of usage for Netflix. This makes **the problem of enforcing DRM over the web an incredibly important problem for Netflix, et al. to solve**. That's where **EMEs come into play**. They are essentially **a technology that allows the browser to communicate with DRM systems over an encrypted medium.** They **make it possible for DRM software to work over the browser and relay encrypted content** to the user. This means **Netflix can serve you a movie that's encrypted and protected by DRM**." *See* Exhibit 6, p. 2, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

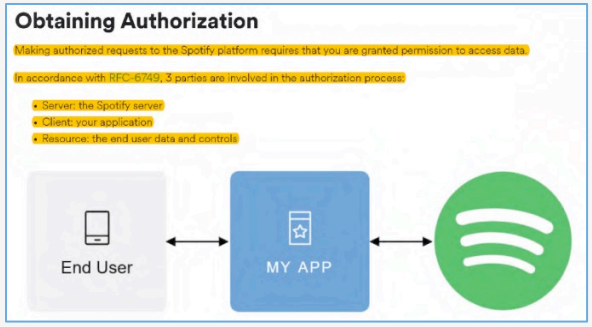
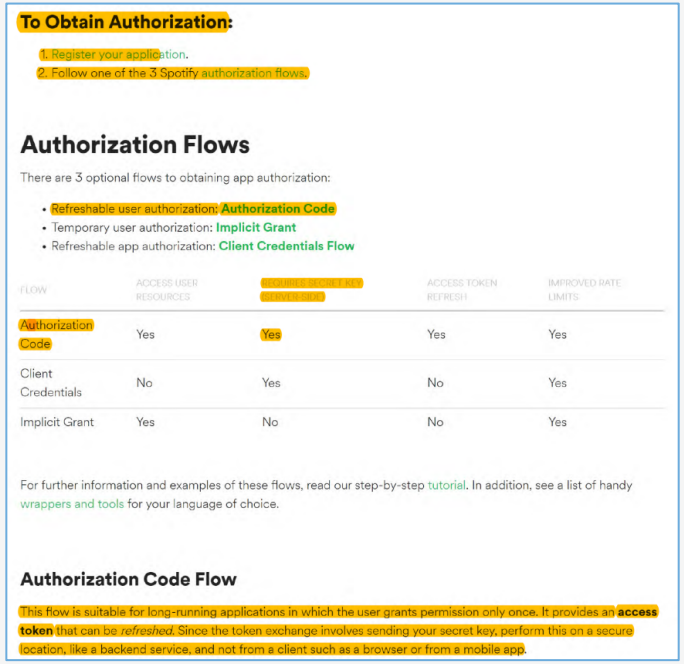
CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	"Say **Felix is a movie streaming service that serves up movies over the browser and uses EMEs to protect the content it delivers**. Suppose that user wants to play some encrypted content from Felix. Here's what happens: **The browser will load up this content, realize it is encrypted and then trigger some Javascript code**. This **Javascript will take the encrypted content and pass it along to something called the Content Decryption Module (CDM)**—more on this later. **The CDM will then trigger a request that will be relayed by Felix to the license server**. This is a Felix-controlled server that determines whether or not a particular user is allowed to play a particular piece of content. **If the license server thinks our user can watch the content, it will send back a decryption key for the content to the CDM**. **The CDM will then decrypt the content and it will begin playing for the user**." The most important **chunk of this whole process is the CDM**. By its nature, the **CDM has to be a piece of code that's trusted by Felix**. If it isn't, Felix can't be sure that it will do what it wants it to. This means that the **CDM has to come with the browser**. There are a couple of different providers for CDMs, including **Google's Widevine** and Adobe Primetime. Both of these provide the **encryption support necessary for content providers such as Felix to enforce DRM**. *See* Exhibit 6, p. 2 & 3, "The most controversial HTML5 extension – LogRocket" blog post (Date Accessed 05/01/2018), available at https://blog.logrocket.com/the-most-controversial-html5-extension-7adc66bbc291 (emphasis added)

CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	**"Encrypted Media Extensions provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video**. EME is designed to **enable the same app and encrypted files to be used in any browser, regardless of the underlying protection system**. The former is made possible by the standardized APIs and flow while the latter is made possible by the concept of Common Encryption EME is an extension to the HTMLMediaElement specification — hence the name. Being an 'extension' means that browser support for EME is optional: if a browser does not support encrypted media, it will not be able to play encrypted media, but EME is not required for HTML spec compliance "EME implementations use the following external components: **Key System: A content protection (DRM) mechanism. EME doesn't define Key Systems themselves, apart from Clear Key (more about that below)**. **Content Decryption Module (CDM): A client-side software or hardware mechanism that enables playback of encrypted media. As with Key Systems, EME doesn't define any CDMs, but provides an interface for applications to interact with CDMs that are available**. **License (Key) server: Interacts with a CDM to provide keys to decrypt media. Negotiation with the license server is the responsibility of the application**. *continued on next slide.*

CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	**Packaging service: Encodes and encrypts media for distribution/consumption**. Note that an application **using EME interacts with a license server to get keys to enable decryption**, but user identity and authentication are not part of EME. **Retrieval of keys to enable media playback happens after (optionally) authenticating a user. Services such as Netflix must authenticate users within their web application**: **when a user signs into the application, the application determines the user's identity and privileges**." *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added) "The **authorization flow we use in this tutorial is the Authorization Code Flow**. **This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token**. The **code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication**." *See* Exhibit 9, p. 1, "Web API Tutorial _ Spotify for Developers" webpage (Date Accessed 05/02/2018), available at https://beta.developer.spotify.com/documentation/web-api/quick-start/ (emphasis added)

CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	"How does EME Work?"  1. A web application attempts to play audio or video that has one or more encrypted streams. 2. The browser recognizes that the media is encrypted (see box below for how that happens) and fires an encrypted event with metadata (initData) obtained from the media about the encryption. 3. The application handles the encrypted event: a. If no MediaKeys object has been associated with the media element, first select an available Key System by using navigator.requestMediaKeySystemAccess() to check what Key Systems are available, then create a MediaKeys object for an available Key System via a MediaKeySystemAccess object. Note that initialization of the MediaKeys object should happen before the first encrypted event. Getting a license server URL is done by the app independently of selecting an available key system. A MediaKeys object represents all the keys available to decrypt the media for an audio or video element. It represents a CDM instance and provides access to the CDM, specifically for creating key sessions, which are used to obtain keys from a license server. b. Once the MediaKeys object has been created, assign it to the media element: setMediaKeys() associates the MediaKeys object with an HTMLMediaElement, so that its keys can be used during playback, i.e. during decoding. 4. The app creates a MediaKeySession by calling createSession() on the MediaKeys. This creates a MediaKeySession, which represents the lifetime of a license and its key(s). 5. The app generates a license request by passing the media data obtained in the encrypted handler to the CDM, by calling generateRequest() on the MediaKeySession. 6. The CDM fires a message event: a request to acquire a key from a license server. 7. The MediaKeySession object receives the message event and the application sends a message to the license server (via XHR, for example). 8. The application receives a response from the license server and passes the data to the CDM using the update() method of the MediaKeySession. 9. The CDM decrypts the media using the keys in the license. A valid key may be used, from any session within the MediaKeys associated with the media element. The CDM will access the key and policy, indexed by Key ID. Media playback resumes. How does the browser know that media is encrypted? This information is in the metadata of the media container file, which will be in a format such as ISO BMFF or WebM. For ISO BMFF this means header metadata, called the protection scheme information box. WebM uses the Matroska ContentEncryption element, with some WebM-specific additions. Guidelines are provided for each container in an EME-specific registry. Note that there may be multiple messages between the CDM and the license server, and all communication in this process is opaque to the browser and application: messages are only understood by the CDM and license server, although the app layer can see what type of message (https://w3c.github.io/encrypted-media/#idl-def-MediaKeyMessageType) the CDM is sending. The license request contains proof of the CDM's validity (and trust relationship) as well as a key to use when encrypting the content key(s) in the resulting license. *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added)

CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	*Upon information and belief, Spotify works similarly utilizing the CDM module as explained here.* "**Getting a key from a license server** In typical commercial use, **content will be encrypted and encoded using a packaging service or tool**. **Once the encrypted media is made available online, a web client can obtain a key (contained within a license) from a license server and use the key to enable decryption and playback of the content**. The following code (adapted from the spec examples) shows **how an application can select an appropriate key system and obtain a key from a license server**."  *See* Exhibit 8, p. 1, "What is EME_ _ Web Fundamentals _ Google Developers" forum page (Date Accessed 05/02/2018), available at https://developers.google.com/web/fundamentals/media/eme (emphasis added)

CLAIM 35	SPOTIFY
wherein the device is configured to steganographically cipher both value-added information and at least one value-added component associated with the value-added information.	  *See* Exhibit 11, p. 1-3, "Authorization Guide _ Spotify for Developers" webpage (Date Accessed 05/03/2018), available at https://beta.developer.spotify.com/documentation/general/guides/authorization-guide/ (emphasis added) *Thus, this claim element is infringed literally, or in the alternative under the doctrine of equivalents.*